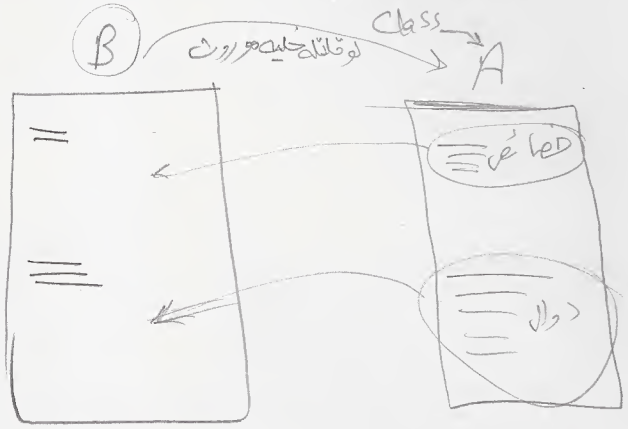
Java 102

lesson #1

الوراثة inheritance

* باختصار لما نقول B ورث من A يعني B أخذ صفات A

* باختصار وراثة تعني " نقل الصفات " .



لو مثلاً كتبت غرض جوه دوال خاصة بـ B وخليت قلنا خلي B يورث من A فذلك يعني : " انقل الخصائص الموجودة في A والدوال والـ actions الموجودة في A انقلها إلى B " . وقت حوم هينة "

* فرضاً أن هناك دالة موجودة في الكلاس A وتسمى Read و B موروث من A

* ثم ذهبت إلى B وعرفت كائن من B

B z = new B();

z.Read(); ← لاحظ أن Read ليست موجودة في B ولكنها موجودة في A وبما أن z كائن من B ، B يرث A فتعتبر Read كأنها موجودة في B .. إذاً بالإمكان استخدامها

* الوراثة توفر عليك إنك تعيد نفس الكلام وكذلك سرعة الإنتاجية

* نفترض أن B كلاس فارغ وبما أن B موروث من A فكده ممكن نقلت الأشياء الموجودة في A إلى B حتى لو لا يوجد دوال

بدون إعادة كتابتها

* إذاً الوراثة هي انتقال الخصائص والـ Actions إلى الكلاس الجديد مع وجود أشياء إضافية

غير ليست جميع الدوال والمتغيرات = دوال متعدم .

lesson 2

إنشاء كلاس

هنعمل كلاس جديد عادي
* لا جديد

---

lesson 3

وعملية الوراثة Extends

(ثم كتابة هذا الكلام عشان تقول له إنه وارث من A)

```
public class B extends A {   // أنشئ كلاس اسمه B ويرث الكلاس A

}
```

//Now:

```
B Z = new B ();
    Z. print A();   // كده هيطبع محتويات دالة print الموجودة في A مع إن B فاضي
```

بس عشان هو وارث من A فيحق له استعمال الدوال والمتغيرات.

---

lesson 4:

أوجه القوة في الوراثة

```
public class B extends A {}
```

معنى الجملة:

* إنه هيأخذ A ويطور عليه "يوسعه" Extend فكأنه B هي تطوير لـ A بدون ما يعدل ع A

* باختصار: B يرث A يعني أنك بتقول أنا بطور A يعني بضيف ع A بس بدون ما أعدل عليها

---

lesson 5

إنشاء الكلاس

* من فوائد الوراثة أيضًا أو أحد أهدافها إنك لو عندك كلاس B مثلاً وعايز تضيف له أكواد

يعني عايز تطوره فبدل ما تعدل في نفس الملف إنت ممكن تعمل ملف جديد وتخليه يرث من هذا الملف

وكده تكمل عليه عادي عشان بقا معاه نفس الصفات والدوال.

* مميزات الوراثة: * في الصيانة * كما تم شرحه

وهذه مميزات قوية لبرمجة الكائنات oop.

lesson6 :

مفهوم subclass & superclass

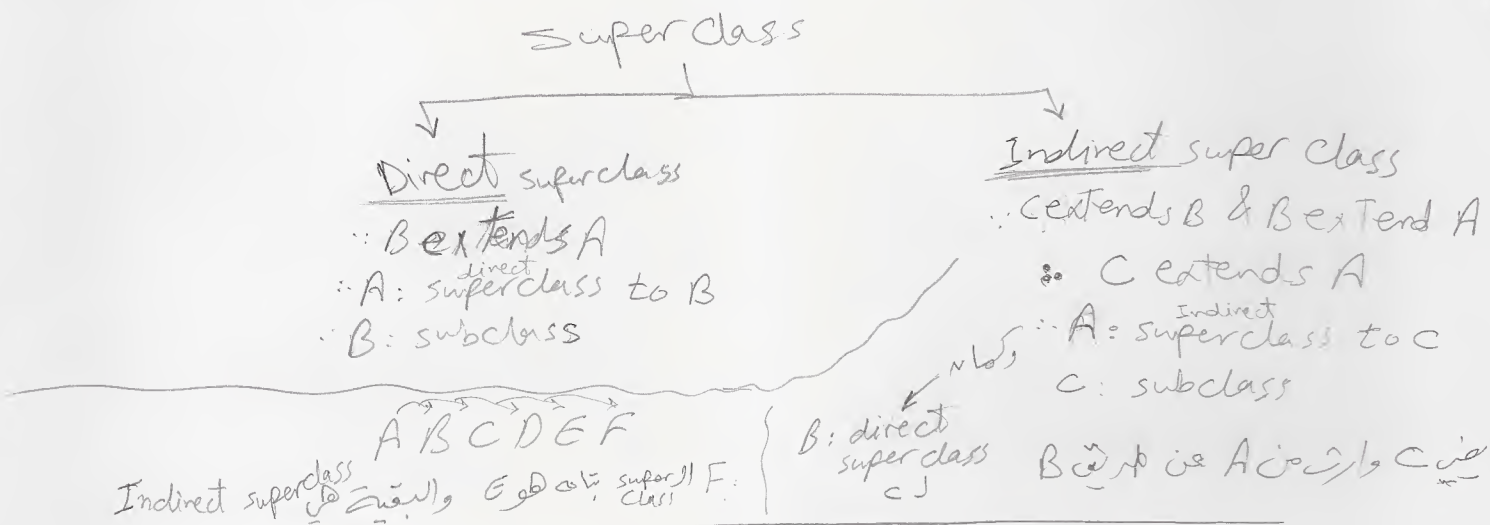
* بما أن الـ B ورث من A إذاً نقدر نقول A : superclass ← الأب

B : subclass ← الابن

---

* بما أن الـ C ورث من B و الـ B ورث من A

فهل إذا الـ A يعتبر superclass لـ C ؟!



يعني C وارث من A عن طريق B وكمان

---

lesson 7 :

التعرف على كلاس object

* يوجد كلاس تلقائي في جافا يسمى object .. بمجرد أن تنشأ أي كلاس في جافا فتلقائياً . كأنك عملت B extends object بدون ما تكتب ^^

* جميع الكلاسات الموجودة في جافا موروثة من كلاس يسمى object

* يعتبر كلاس object هو الأب " لا يوجد أعلى منه "

* أي superclass موجود هو الكلاس object

---

lesson 8

التعرف على super

* تعني كلمة super : الكلاس الموروث منه

* تمكنك super من الوصول إلى الأشياء الموجودة في الكلاس الأب superclass

```
public class B extends A{
public void printB(){
System.out.println("classB");
super.printA();
}
}
```

A

lesson9

Constractor ، دالة الـ super

lesson10

* دوال الـ Constructor لازم تورثها بل عموم أنت باستدعائها مع استخدام super !

Ex

class A

```
public class A {
    public int val;
    public A(int c){
        this.val = c
    }
    .
    .
    .
}
```

Class B

```
public class B extends A {
    public B()
    {
        super(0);
    }
}
```

lesson11

معلومات أكثر حول الـ Constructor

```
A x = new A();
A y = new A();
```

↳ Constructor بعد كلمة new تأتي دالة الـ للتعريف

```
x ↵
    this.val = 5    =   x.val = 5
    this.val = 10   =   y.val = 10
y ↵
```

lesson12

استخدام أكثر من Constructor

* بالإمكان استخدام أكثر من Constructor ولكن بشرط ألا يتشابهوا في الوظيفة أو النوع "أو الباراميتر".

Ex :-

```
public int val;
public A()
{ this.val=10; }

public A(int c)
{ this.val=c; }

public A(int a, int b)
{ this.val= a*b; }
```

~~public A(int s)~~

مينفعش عشان هناك Constructor آخر بنفس الوظيفة وهو

✓

Lesson 13

الـ Constructor وعملية الوراثة

```
B x = new B();
```

↳ معناها : استدعي الـ .Const بتاع الـ B

* لو عندك class A و class B مثلاً :

A.Java

```
public class A {
 public int val;
 public A()
 {  This.val=10;
 }
}
```

B.Java

```
public class B extends A {
  public B()
  {   // A(); كأنه استدعى
   this.val=500;
  }
```

يعني تسلسل التنفيذ هتكون val = 10 وبعد كدة val = 500

Lesson 14

super و Constructor

```
public class B extends A{
 public B()
 {
   super(70);   // super بتاع الـ Constructor كده انت بتقله استدعي الـ
                // والذي له باراميتر واحد
 }
}
```

super ← دا هوا A

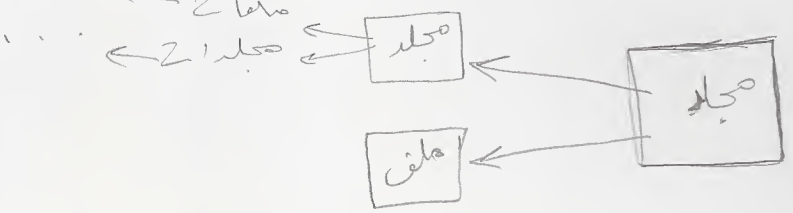
↳ وممكن يكون ٢ باراميتر ٣... أي عدد
ع حسب البيانات كاملة.

Lesson 15

استخدام الـ package

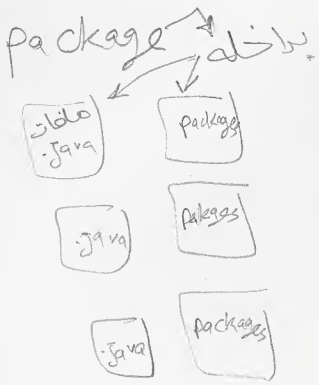
* بما أننا منظمة عن طريق أشياء تسمى packages .

* في نظم التشغيل



* جاوا منظمة بنفس التنظيم أعلاه .

في "مجلد" يعمل عمل "مجلد"



Ex: package x

بداخلها مجموعة كلاسات ومجموعة packages

ومجموعة الـ packages بداخلها أيضاً كلاسات وpackages أخرى

وهكذا !

* الطريقة هذه طريقة تنظيمية فقط ما بيهمه لأسلوب المجلدات .

* تساعد في تنظيم الشغل والوصول لأي كلاس تريده .

* ملاحظة هامة : لأنه عندما تريد استخدام شيء معين لازم تحدد له فين موجود في أي package عشان توصلوا وتتعامل معاه مرتقدر تستخدموا .

Lesson 16

مفهوم الـ Access Modifiers

* public
* protected
* private
* :

* إذا وضعت الـ Access Modifiers قبل المتغيرات أو الدوال فهي تحدد طريقة الوصول لهذا المتغير أو الدالة .

* إذاً باختصار الـ A.M "تتحكم في الوصول" . "هل تقدر توصل للمتغير ده؟ أو الدالة دي؟"

public : وصول عام

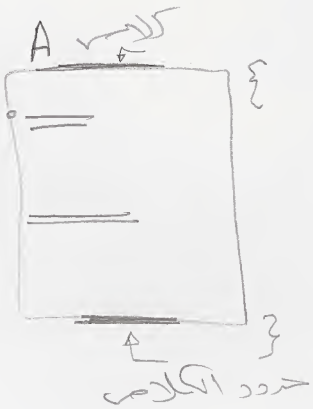
protected : وصول محمي

private : وصول خاص

Lesson 17

private حـ

private : خاص



⇒ private int x ;

معناه أن المتغير x لا يستخدم خارج الكلاس
فهو خاص بالكلاس ولا يستخدم خارج حدود الكلاس

Ex: A b = new A ();

b.(x) = 10 ; (XXX)

خطأ لأنه x هنا تخدم خارج حدود الكلاس
لأن الكائن قام باخراج الـ x خارج الكلاس

Lesson 18

استخدام private

```
public class A {
private int val;
public A()
{ This.val =10;
 }
}
```

/* val استخدام private وان المقصود أو هدف
لن يخرج خارج القوسين */

* وظيفة private : هي الحفاظ على البيانات.

lesson 19

استخدام set , get مع private

كيف نتعامل مع المتغيرات المعرفة private - من الخارج

عن طريق setter و getter . "عبارة عن دوال" — يُرسل قيمة

* هي عبارة عن دوال "جمل" إنت بتكتبها عشان ترسل أو تطلب قيم المتغير الخاص private

Ex:
```
public class A{
private int val;
public void setVal (int c).
  { this.val = c;
  }
public int getVal()
  { return this.val;
  }
}
```

set: دالة تُعين قيمة لمتغير نوعه private

get: دالة عشان نقدر نستخدم قيمة المتغير اللي نوعه private

* set , get تُستخدم كطرق بين البرمجيين

---

lesson 20

فوائد / استخدام private

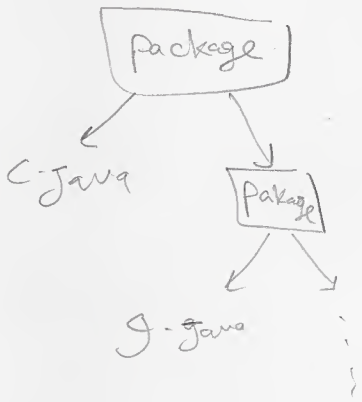
* إخفاء البيانات
* حماية وإمانة للبيانات بحيث لا تستدعيها بشكل غير مباشر
* تحكم كامل
* سرعة التعديل والصيانة للكود.

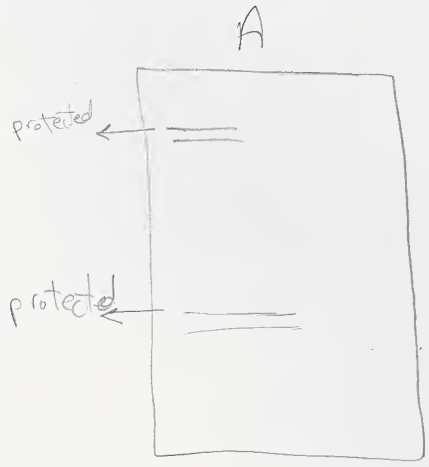
---

lesson 21

شرح protected

محمي





protected : محمي من أي كلاس أو أي مكان باستثناء حالتين:-

① أي كلاس مشتق من A ← يعني أي subclass مهما كان

② جميع الكلاسات التي في نفس الـ package الموجود بـ A

③ في نفس الكلاس المستخدم فيه

---

lesson 22

استخدام protected

شاهد الفيديو

---

lesson 23

استخدام public

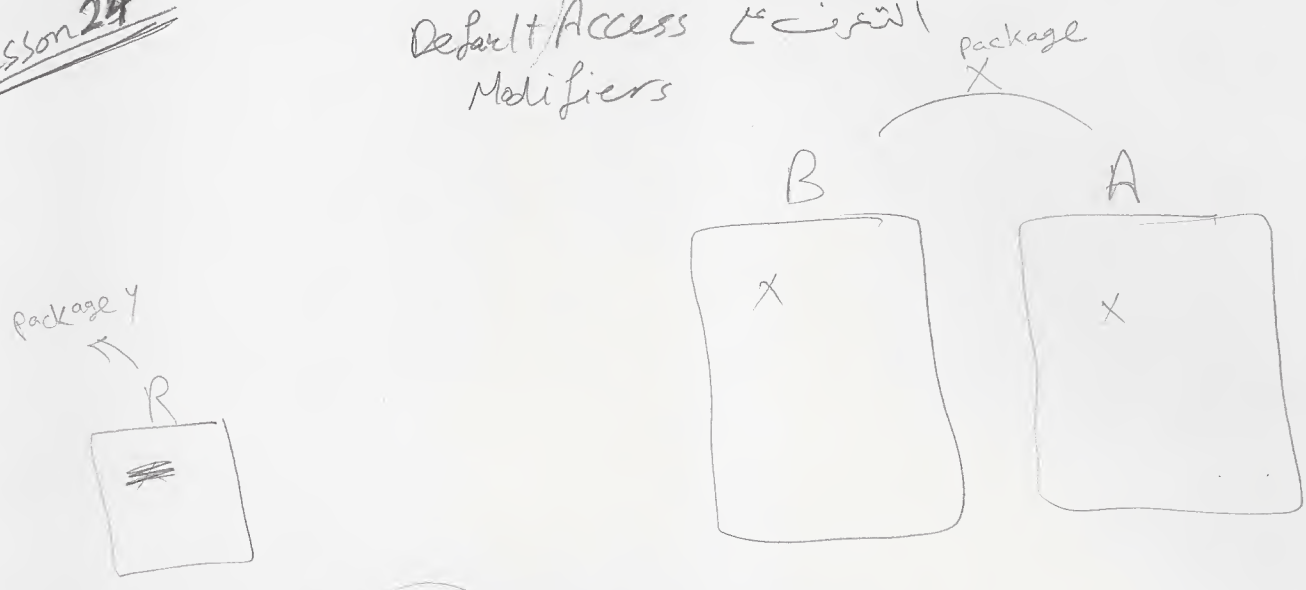
* وتعني عام وبإختصار تستخدم في أي مكان

* لا ينصح public للمتغيرات مع أنها لا تسبب أي مشكلات «جسدنا بتتعارض مع مفهوم برمجة الكائنات»

* ينصح بجعل المتغيرات private أو protected واستخدم set, get طلب وتخزين البيانات

lesson 24

Default/Access التعرف على Modifiers



public protected private ← لم تحدد هنا int x ;

إذاً هنا يسمى Default

⇐ وظيفة هذا Access Modifier أن x مش هيقدر يوصلوا إليه عن طريق الكلاسات الموجودة في نفس الـ Package

⇐ يعني كلاس R لن يقدر يوصل للمتغير x

---

lesson 25

استخدام Default Access Mod.

نصيحة: (يستخدم المتغيرات مع كل private أو protected "في الغالب" واستخدم get, set

---

Simple Calc.

lesson 26

lesson 27

* شاهد الفيديو

lesson 28

---

another Calc.

lesson 29

lesson 30

* شاهد الفيديو

* استخدم نوع البيانات double عشان العملية تشتغل كويس

lesson 31

التعرف على static attribute

لو كتبت عند x هذا بعد كلمة public / private ⟸ static

ex : public static int x ;

* معناها: أن المتغير x مشترك بين جميع الكائنات التي تُنتج من الكلاس A

أعضاء الكلاس Class Member ويتم تسميتها "متغير كلاس".

* إذا كرة الـ static معناه إنه لا يحتاج الكائن للتعريف أو استخدامه

يعني لما تريد الوصول لـ x ، طالما أنه static فلن تحتاج كائن

* فكرة أي كائن جديد له x واحدة فقط، المستخدمة في جميع الكائنات

* يعني فيه x لو تغيرت في أي كائن ⟸ سوف تتغير في البقية.

* فأصبح الـ static هو شيء مشترك بين جميع الكائنات ولا يوجد منه إلا نسخة واحدة فقط.

* للوصول للمتغير من الخارج يتم كتابة: A.X = 10; ← اسم الكلاس ← اسم المتغير static

* الـ Member التي من نوع static لا تحتاج لإنشاء كائن "لأنها ليست خاصة بكائن معين بل مشترك"

lesson 32 — استخدام static

```
package java102;
public class A {
    public int x; // متغير ينتج لجميع الكائنات نسخة
    public static int y; // متغير مفرد لا نسخة واحدة منه // لجميع الكائنات //
{
```

وللوصول لهذا المتغير الـ static ← يتم كتابة:

```
A.y = 77;
```

A ← اسم الكلاس
y ← المتغير الذي نوعه static

* فائدة:- معرفة عدد الكائنات.
بواسطة إنشاء متغير يعد الكائنات التي تنتج من الكلاس A .

* أيضاً لا تحتاج إنشاء كائن للوصول للمتغير وذلك لأنه ظاهر بالكلاس وليس بالكائن

---

lesson 33 — مفهوم Static Method

```
public static void Example()
{
    System.out.println("Hi...");
}
```

* نفس الكتابة في الدرس السابق
* هذه الدالة لا تستدعى عن طريق كائن
بل عن طريق اسم الكلاس واسم الدالة فقط

* الدالة إذا كانت static معناها ظاهرة بالكلاس وليس الكائنات

Ex: A.Example();

* قاعدة هامة :- static لا يستخدم بداخلها this "السبب في الفيديو" توضيح

لأنه لم يتم إنشاء كائن

* دوال الـ static تستخدم بدون إنشاء كائن.

lesson 34 — مثال ع Static Method

---

lesson 35 — تلخيص موضوع static

```
public int x ;
public static int y = 10 ;
```

* أي دالة أو Method أو procedures أو أي حاجة نوعها static ⇐ فلا تقدر تتعامل مع المتغيرات التي بداخل الكلاس ونوعها ليس static . والعكس صحيح . "لأن اليمين غير static هينسخ عادي ، بينما الي ب static لن يتم نسخه ".

* لو تم وضع private ~~protected~~ قبل static فلن تقدر توصل لها خارج الكلاس إلا عن طريق الحيلة get & set

* static يعني تقدر تستدعي بدون إنشاء كائن يعني ممكن تستدعي عن طريق اسم الكلاس

* لا يتم إستخدام this مع static أبدا

---

lesson 36 — مثال TheCount

```
public class TheCount {
 public static int Count = 0 ;
  public TheCount ()
   { Count ++ ;
   }
}
```

معنى الدالة :- كل ما أنشئ كائن ⇐ يزيد ال Count بواحد

lesson 37 — تابع المثال

lesson 38

مفهوم الـ Method overloading

أو

المقصود بها : تقدر في نفس الكلاس إنك تضع دالة x بنفس الإسم بإختلاف البارامترات في النوع أو العدد وممكن في الـ return type .

lesson 39

مثال Method overloading

شاهد الفيديو

lesson 40

مفهوم Method Overriding

المقصود بها : إلغاء الدالة المستخدمة في superclass سابقاً واستخدام مكانها الدالة الجديدة في الـ subclass .

Class A

```
public int beans(int value)
{ return 0; }
```

Class B

```
public class B extends A {
@Override
public int beans (int val)
{ return 1; }
```

* فائدتها هي تطوير الدالة بعد ما ورثتها .
* تستخدم في حالة الوراثة من كلاس « superclass »

مقارنة

lesson 41

Method overriding	Method overloading
* يوجد بـ super class دالة اسمها زيها x فإذا أكدت كتابة x في الـ subclass فنفس العملية overriding	* إذا كتبت الـ Method واختلفت في البارامترات فتعتبر الدالة overloading (بنفس الاسم "prototype" Signature)
* كأنك بتقول إلغاء الدالة الموجودة في الـ superclass واستخدم الدالة الموجودة في الـ subclass	* تستخدم في نفس الكلاس أو كلاسات مشتقة "موروثة" والمهم يختلفوا في البارامترات عددها أو أنواعها
* مكرتة بإعادة كتابة نفس الدالة الموروثة	* لا يهتم بنوع الإرجاع
* تستخدم في الوراثة .	

lesson 42 "overloading" مثال overCalc

شاهد الفيديو

---

lesson 43 مثال ع عمل overriding للطباعة

---

lesson 44 استخدام final

التعامل مع الثوابت

* احنا عندنا متغيرات وثوابت.

مثال ع المتغير: public int x=0;

ثابت زي π مثلاً π=3.14 ده ثابت فيزيائي ويوجد ثوابت أخرى في فيزياء هنشوفها في البرنامج.

مثال ع الثوابت ← طيب public final int y;

final ← ثابت لا تتغير قيمته

يوجد مكانين للتعامل أو وضع قيمة الثابت:

① public final int y=10; في مكان التعريف

② في أثناء الإنشاء

وكده القيمة مش هتتغير

① وضع قيمة الثابت داخل الـ Constructor أثناء تعريف الكائن

② وقت إنشاء الكائن عبر الـ Constructor.

⇐ الـ final يتم وضع قيمته مرة واحدة فقط وإذا تم وضعها لا تتغير.

---

lesson 45 استخدام final static

إذا كان الثابت لن تتغير قيمته بين الكائنات فنضع بوضع final static

باختصار هالشي:

>> إذا كان كل كائن له ثابت معين فحنا لا نضع static

>> إذا كل الكائنات التي سننشئها لها نفس قيمة الثابت فنضع وضع static لأن الثابت سيكون مشترك بين كل الكائنات. وهنا لا يتم إنشاؤه إلا مرة واحدة في الذاكرة.

lesson 46

مثال على استخدام final static

---

lesson 47

A

التعرف على Final Method

معنى كلمة final إذا جاءت مع الـ Method

```
public final int read()
{ return 0; }
```

⇐ ممنوع عمل override لهذه الـ Method

يعني أي كلاس يورث من A لازم يستخدم الدالة read كما هي بدون ما يعدل عليها override

يعني لما نحط final كأنك بتقول هذه الدالة لا تحتاج لتعديل.

⇐ استخدامها لو عايز تخلي الدالة تجمع الكلمات الموروثة ومش عايز أي كلاس يعدل عليها

---

lesson 48

التعرف على final class

معنى final إذا جاءت قبل الـ class:

```
public final class A{}
```

⇐ ممنوع يُورث منه أي كلاس ومستحيل.

"يعني ما يحتاج تعديل أو تطوير"

⇐ يعني بكلام آخر الـ final تقوم بإيقاف عمل الـ extends

---

lesson 49

مثال على final Method

خالد الغبيري ..

lesson 50

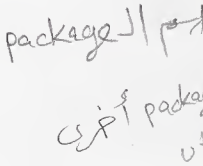
استخدام import

* جافا تكون محمل فيها عدد كبير من الكلاسات الجاهزة المكتوبة مسبقاً .
* يتم استدعاء هذه الكلاسات عن طريق import .
* لإستدعاء هذه الكلاسات يتم كتابة import بعد package " يتم الاستدعاء أولاً »

import java.

* Ex :- import java.util.Date; == Date استدعى الكلاس الذي اسمه

كذلك كتب مسار



package java » package util » class Date

* import تستخدم لإستيراد كلاسات جاهزة للتعامل معي .
* لإ ستيراد جميع الكلاسات داخل الـ package يتم كتابة (*) مثال: import java.util.*;

, لكن لا يفضل لأنه يكبر حجم البرنامج

---

lesson 51

عمل import Math

* كلاس للتعامل مع العمليات الرياضية import java.lang.Math
* يمكن تستخدم الـ package lang بدون عمل import لأنها اختيارية موجودة أصلاً .
* معنى الـ system.out اللي كنا بنستخدمها مثل موجودة في lang .

---

lesson 52

استخدام كلاس Scanner

* لإ دخال القيم بنفسك من المستخدم يتم استخدام كلاس scanner
* لإستدعاء هذا الكلاس import java.util.Scanner;

* بعد ما عملت import java.util.Scanner;

* ثم استدعاء الكلاس:

- Scanner f = new Scanner(system.in);

انشئ كائن جديد "الذي سيقرأ القيم" واسمه f

ويأخذ بارامتر، وهذا البارامتر هو كائن منبع المدخلات "sytem.in"

* system.out ⇒ printstream ⟵ يمكنه يمثل الشاشة
* system.in ⇒ Input stream ⟵ يمكنه الشيء الذي يدخل للشاشة

"System.in"

* in ⇒ هيمثل المدخلات عبر الشاشة
* out ⇒ المخرجات عبر الشاشة

---

* كود كامل دوال - "Method" - الكلاس Scanner

يمكن int

بعد ما نقوله دالة تسمى: nextInt(); ⇐ تعني الانتظار للذي سيقوم باستخدامها

Ex: Scanner f = new scanner (system.in);

system.out.println("Please Enter your age:");

int c = f.nextInt(); // أول ما المستخدم يدخل القيمة سيتم وضعها داخل c

---

Lesson 53 — قراءة قيمة نصية باستخدام scanner

Scanner f = new scanner (system.in);

String name;

system.out.println("please enter your name:");

name = f.next(); // string طالما من دوال scanner فإنها نصية وسترجعها كـ

system.out.println("your name is:" + name);

lesson 54

التعديل على كلاس simple calc

* يمكن تعريف الكلاس scanner كمتغير "لأنه نوع بيانات"

ex: private Scanner scn ;

سيتم التحدث عنها بعد كده بالتفصيل

* شاهد الفيديو .

---

lesson 55

مفهوم الـ Composition

* لماذا تنشئ الكلاس ؟

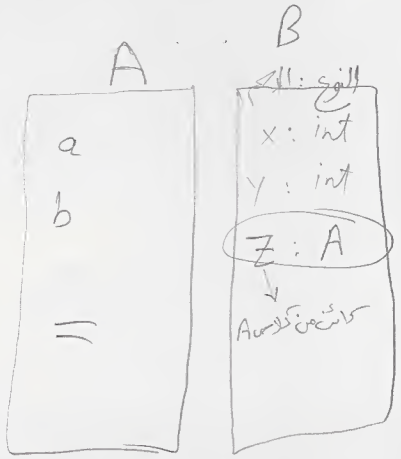
* أحيانا إنت لما تنشأ الكلاس ⇐ إذا أنت بتنشأ نوع بيانات جديد .

Java defined

Ex: int x ; // عرفت متغير نوعه Integer

Ex: A x ; // عرفت متغير نوعه A كائن

user defined



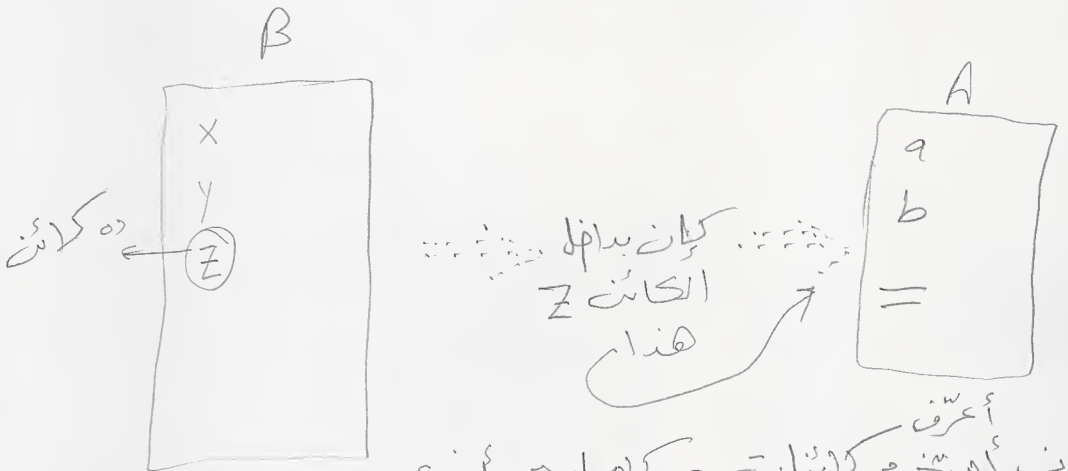
⇐ معنى الـ Composition : إنك تعرف كائن ⇐ هذا الكائن من كلاس آخر "ليس من الأنواع الافتراضية Ex: int, long, short..

⇐ باختصار : معنى الـ Composition : كائنات تدخل في تعريف كلاس آخر، يعني ربط .

A Z ; // كده عرفت كائن من الكلاس A

Z ← Attribute يعتبر كده

كائن من class A



⇐ معنى الـ Composition : إني أستخدم كائنات من كلاسات أخرى أعرّف كمتغيرات "attributes" في كلاس جديد .

lesson 56 — مثال على مفهوم Composition

* بمجرد إنك عرفت object من كلاس آخر ← داخل الكلاس ← فوذا يسمى Composition

مثال: أي عربية ← كده هقدر آخد خدمات الكلاس scanner // private Scanner scr ;

* أحيانا لما نعرف كلاس جديد ← كده إنت بتعرف نوع جديد في اللغة ← عشان كده أقدر أستخدمه كخاصية.

⇐ ملخص:

* الـ Composition: هو عبارة عن وضع كائنات من كلاسات أخرى في تعريف كلاس جديد.

* فائدته: بناء كلاسات معقدة بسهولة وسهولة التعديل.

---

lesson 57 — مقدمة للاستثناءات Exceptions

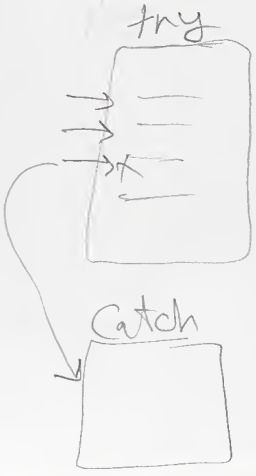
* Exceptions = معالجة الأخطاء

* Exceptions: هو أسلوب لمعالجة الأخطاء

وهذا الكود في الحالية إنه يحصل له خطأ

مثال: إنت بتبرمج الكود إنه يتعامل بالنت، المستخدم داخل النت هنا ممكن يقع في خطأ. "Exceptions"

* Exceptions: يعني غير المعالج لسير الكود «غير التحكم بالكود»



⇐ أزاي نحمي كود ممكن يقع في مشكلة

كده لازم نحط الكود داخل بلوك اسمه try ← يعني الكود اللي

⇐ إنت لما حطيت الكود داخل بلوك try كده إنت عطيت كود مشكوك في أمره يعني ربما يحدث استثناء "خطأ"

⇐ Catch: إذا حدث خطأ في بلوك try هيعمل jump ويروح لـ Catch

ولو مفيش خطأ: فلازم مش هيدخل لـ Catch

* يعني في البلوك catch إنت هتعمله إزاي يعالج الخطأ

Lesson 58 :- مثال ع الاستثناءات

Lesson 59 استخدام try & catch

* يتم وضع الكود المشبوه داخل try { } ← يعني اللي فيه احتمالية خطأ

* يجب استخدام catch أيضاً طالما استخدمت try

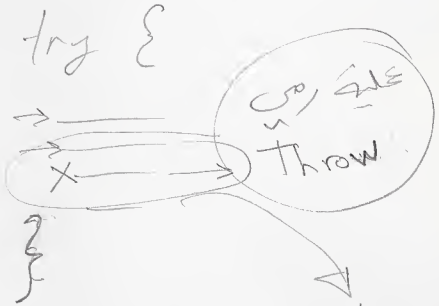
```
try {
  ...
}
catch(Exception e)
{
  ...
}
```

(Exception ← الاسم الخطأ، e ← الاسم)

* ميزة الـ Exception إنها تعطيك فرصة قبل إنهاء البرنامج

يعني قبل إيقاف البرنامج "يعني لو حصل استثناء ← البرنامج هيقف"

Lesson 60 الكائن المستخدم مع Catch



* عندما يحدث خطأ ← ترسل بيانات الخطأ لكائن يسمى Exception الذي قمت بتعريفه بين قوسين Catch

* المعلومات الخاصة بالخطأ الذي وقع يتم تخزينها بالكائن الذي انشئته من الكلاس Exception الي هما e ←

e ← اسم كائن من نوع Exception

lesson 61

مثال لإستخدام كائن catch

* شاهد الفيديو

* Exception هي إستثناء عام "جميع كلاسات الاستثناءات مشتقة منه".

* يفيدنا هذا السطر في معرفة نوع الاستثناء

output : Java.util.InputMismatchException

اسم كلاس الإستثناء

* يعني لو جيت غيرت نوع الكلاس في القوس

```
catch(InputMismatchException e)
{
  -
  -
}
```

ولكن هذا الكود بداخل الـ catch لن يتم تنفيذه إلا إذا حدث خطأ من النوع InputMismatchException

---

lesson 62

استخدام أكثر من catch

بإمكانك أضف كود catch مرة أخرى

```
catch(){..}
catch(){}
```

* يمكنك وضع عدد لامحدود من الـ catch على حسب احتمالات الأخطاء اللي ممكن.

---

lesson 63

استخدام catch واحدة للجميع

* وذلك باستخدام الكلاس Exception لأن باقي الكلاسات بتاعة الاستثناءات مشتقة منه

```
try {..}
catch(Exception e) {..}
```

lesson 64

استخدام empty Catch

* معناها: "لا أريد أن أتخذ أي إجراء ← وكمل البرنامج عادي" "Ignore it"

* الـ Catch الفارغ هو عبارة عن عدم إتخاذ أي إجراء مع الخطأ.

---

lesson 65

التعرف على Finally

```
try {
  ...
}
catch(Exception e){
  ...
}
finally{
  ...
}
```

* Finally ← يأتي بآخر الـ Exception

* في وجود استثناء أو عدم وجود استثناء ستنفذ الـ Finally

* فائدته ← تنظيف الذاكرة / عن الموارد التي استخدمتها
← تنظيف البرنامج من الملفات التي استخدمها ولم يستفيد منها

* تفكرته: إنقاذ ما يمكن إنقاذه

---

lesson 66

التعامل مع Finally و Return

```
public static void main(String[] arg){
  : openNet....
  :
  :
  :
  catch(Exception e){
    return;   // main عن الخروج من
  }
  finally {
    .....
  }
  system.out....
}
```

Return; "خروج من الدالة"

عن مستخدمة "عن لو الجهاز مكانه بالذاكرة هيكون له محجوز"
بس كده الحجات لسه مفتوحة

عشان كده دي يوضح قوة استخدام Finally

لن ينفذ هذا الكود - عن هينفذ Finally وثم Return
لو وجدت Return هينفذه.

lesson 67 — التعرف على throw

* بسبب أن الدالة nextInt() ترمي استثناء لذلك وضعناها داخل بلوك try
يعني هي عارفة scanner كلاس معرف من قبل جافا .

* يعني لازم تعمل دالة ← ترمي استثناء

```
package java102

public class A{
  public void fun(int val)
    if(val>100)
    {
      throw new Exception("Value overflow");
    }
```

throw يعني ارمي استثناء

الكود ناقص بس تابع ← الكود بعرف كائن استثناءات

* كلمة throw لازم بأي بعدها كائن "الكائن الذي سيُرمى"

lesson 68:- التعرف على Exception catcher

* أي دالة ترمي استثناء تسمى Exception thrower
* وأي دالة ترمي استثناء يجب كتابتها داخل بلوك try (أو داخل استدعائها)

"أنواع دوال الاستثناءات" Exception thrower

- exception catcher
- exception propagator

معناه: الدالة ترمي الاستثناء وتعالجه بنفسها

يعني throw تحدد داخل بلوك try

```
try{ if(val>100)
     { throw new Exception("Error");
     }}
catch(Exception e){ }
```

هيستقبل الاستثناء الذي رُمي من هناك

المثال القادم

lesson 69

التعرف على Exception propagator

* Exception propagator: يعني استثناء ولا تعالجه الدالة يرميه

```
public void Fun(int val) throws Exception
{
    if(val > 100)
    {
        throw new Exception("Error");
    }
}
```

* throws Exception: يعني تحتاج إنت تعالجه في أي مكان آخر وليس لازم في نفس الدالة
* ليس propagator لأن الاستثناء خرج من الدالة وانتشر وراح لكود آخر "الي هو مكان الاستدعاء"
  ↳ والكود هو مسؤل انه يعمله catch داخل البلوك...
* propagate: يعني الاستثناء رمي ولكن لم يكتمل ↓ يتم معالجته
* تتم المعالجة عن طريق لما تستدعي الدالة تحطها الدالة في بلوك try, catch في الـ main.

يعني هذه الدالة ترمي استثناء ونوعه propagator

وبالإمكان كتابة أكثر من نوع ولكن يجب عمل import للكلاس

لأن كلاس الـ Exception افتراضي ولا يحتاج عمل import

Ex: `public void Fun(int val) throws Exception, InputMismatchException`

يعني دالة Fun هترمي واحد من الاستثناءات التالية

يعني لما تيجي تعمل catch ⇐ هتعمله لأي كلاس من الكلاسين

---

lesson 70

مثال عن Exception catcher

---

lesson 71

مثال عن Exception propagator

Lesson 72

propagator و Runtime Exception نقطة مهمة جدا

* Runtime Exception ← استثناء لا يحدث إلا وقت التشغيل مثل InputMismatchException

لأن الـ Input تحتاج تشغيل لتأخذ منك قيمة

* Runtime Exception ← لا تقدر تتعرف فيه إلا بعد تشغيل البرنامج

```
import InputMismatchException ;
public class A {
    public void setMaxValue(int val) throws InputMismatchException
    { if (val > 100)
        throw new InputMismatchException ("Error");
      System.out.println(val);
    }
}
```

لو شلت هذا السطر مش هيحصل error "يعني مش هيحصل حاجة"

* إذا كان نوع الـ Exception هو Runtime ← دالة الـ propagator يكون مع النوع هذا اختياري.

* يعني لما تلاقي كدة فاعرف إن الاستثناء ده يحصل وقت التشغيل Runtime.

Lesson 73

Checked & unchecked Exceptions

Exceptions ← "أنواع الاستثناءات نفسها"

Checked

* إذا كان الاستثناء يُشيك عليه أثناء الـ Compile time

~~وقت تشغيل البرنامج~~
أثناء كتابة البرنامج

* يعني يتأكد من الاستثناء قبل التشغيل
* يكشف المشاكل قبل تشغيل البرنامج
* يفحص قبل التشغيل

un checked

* يحدث Runtime Exception

* يعني هذا النوع لن يتم التشيك عليه إلا وقت التنفيذ لأن أنا لا أعرف هذه استثناء أم لم يحدث
* مثال: القسمة على صفر. هذا استثناء لم نشيك عليه وقت الـ Compile
* يكشف المشاكل بعد تشغيل البرنامج
* يفحص بعد التشغيل

lesson 74

أنواع كلاسات الـ Exception

www.docs.oracle.com/javase/7/docs/api/index.html

---

lesson 75

التعامل مع java.lang

* java.lang.* : يعني لو استخدمت أي كلاس منها ⇐ مايحتاج إنك تعمل import

لأن دي حاجات اختراعية في اللغة " language "

* java.lang : عبارة عن package اختراعية.

---

lesson 76

إنشاء كلاس MyException

* أول حاجة اعمل كلاس عادي عادي
* بعدكده خلي الكلاس ده يرث من كلاس Exception ⇐ extends Exception
- الكلاس MyException الكونستركتور بتاعه بياخد رسالة خطأ MyException("Error")

---

lesson 77

تابع المثال

---

lesson 78

التعرف على assertion

* تذكر assert تستخدم في أي Expression، وهذا الـ Expression يعطي true أو false
* Expression زي مثلا لو x>y ... الخ.
* نتيجة الـ Expression لازم تكون true أو false و true بتعادل false بتعادل AssertionError
* فكرة assert هي التعامل مع الأخطاء المنطقية Logical Errors

Ex:
```
int x, y;
x = 10;
y = 5;
assert x>y ;   // تأكد إن x أكبر من y
```
المعنى

* لو رجعت false "التأكد طلع غلط" سيتم رمي استثناء يسمى AssertionError
* يجب تفعيل الـ assert لأنها تحتاج تفعيل

Run » set project Config. » Customize » vm » -ea

vm = virtual machine

يعني مش بتشتغل تلقائيًا

lesson 79

التعرف على صيغة أخرى لـ assert

الرسالة إيجازها

assert x>y : "x less than y";

⇐ بين لما يحدث خطأ استثناء هيظهر وصف للخطأ مكتوب فيه xlessthany .

---

lesson 80

مقارنة بين If , assert

* بالإمكان تقريباً معاملتها معاملة If ولكن للأخطاء

* معنى في المثال لو لم يتحقق الشرط سيحدث خطأ "Error" "استثناء"

بينما If مفيهاش أخطاء ex: assert ((x<=10) && (y<=100)) : "Error";

إذا لم يتحقق الشرط هيظهر الرسالة

---

lesson 81

كلام جداً جداً جداً مفهوم الـ polymorphism

* ترجمتها : تعدد الأشكال

lesson 82

* polymorphism ⇐ حين تستدعي الدالة بدون معرفة الكائن الذي ستستدعى منه

⇐ حين هتستدعي دالة والكائن مجهول

```
public void poly(A value)
{
    value.print();
}
```

value كائن من الكلاس A أو أحد مشتقاته

* الدالة واحدة لكن الذي يتغير هو الكائنات

lesson 83

* الدالة تقدر تنفذها بأكثر من شكل ع حسب الكائن الذي ستكتبه في البارامتر وقت التنفيذ

* معنى آخر لتعدد الأشكال ⇐ الدالة زي ما هي والذي يتغير هو الكائن وبالتالي ستتغير طريقة التنفيذ بناءً ع الكود الموجود في الكائنات المختلفة المستخدمة

lesson 84

مثال

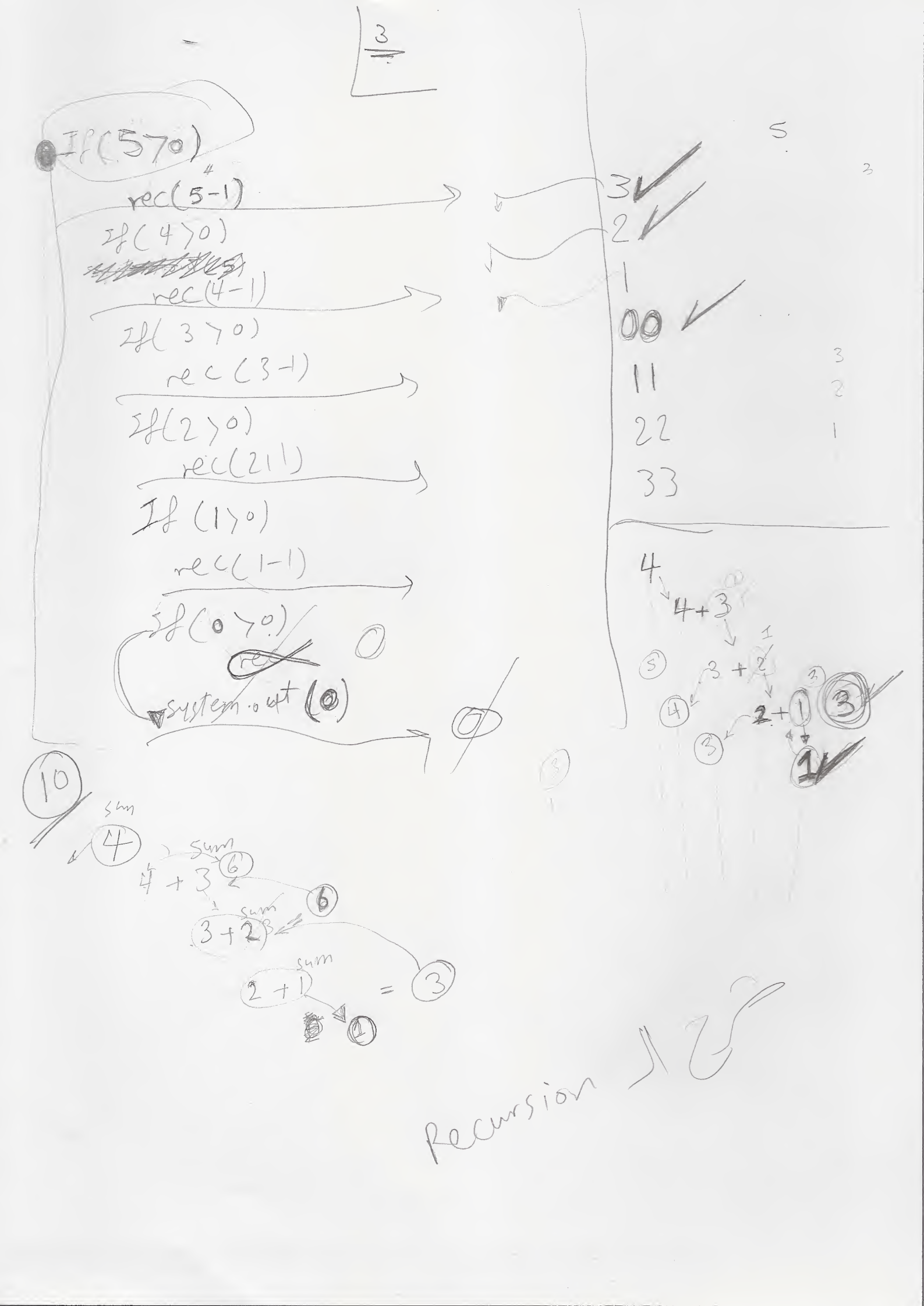
lesson 85

* في الـ polymorphism الكلاس الأساسي لازم يوفر الدالة ع أساس الكلاسات الأخرى تعمل لها override بما يوافقها

lesson 86:

* إذا أنشأت دالة جديدة وصدرت الكلاس الأب ⇐ تقدر تطبق عليها حركة الـ Polymorphism

lesson 87:


3
If(5>0)
rec(5-1)
If(4>0)
rec(4-1)
If(3>0)
rec(3-1)
If(2>0)
rec(2-1)
If(1>0)
rec(1-1)
If(0>0)
System.out(0)
3
2
1
00
11
22
33
4
4+3
3+2
2+1
10
sum
4+3
3+2
2+1
= 3
Recursion

lesson 88

شرح Recursive Method

⇐ طريقة إبهارك بيس

⇐ هي عبارة عن دالة تستدعي نفسها ⇐ يعني بداخل نفس التعريف يوجد استدعاء لنفس الدالة

⇐ يجب وضع حالة توقف لل Recursive حتى لا يحدث خطأ.

lesson 89

مثال

فائدتها: * التعامل مع ال tree

* " " البيانات المتداخلة

* محاولة تكرار عمليات

* المعرب ! ---- الخ

* شاهد الفيديو

lesson 90

كلاس Student

* int: نوع بيانات معرف في الفا مسبقاً ويحتفظ بقيمة واحدة فقط.

lesson 91

* الكلاس يكون بداخله مجموعة بيانات مختلفة داخل الكلاس "Encapsulation" التغليف

lesson 92

* كتابة الدوال هكذا: setName ← Capital, set ← small

Lesson 93

تسمى Camel style مثال الجمل

* يوجد ٣ أنواع من الستايلات:

* C-style ← مثال set_name() كل الكلمات صغيرة وبينها _

* pascal-style ← SetName() كل الكلمات Capital

* Camel style ← setName

small ← set

Capital ← Name

lesson 94
" 95
" 96
" 97
" 98
" 99

lesson 100 — كلاس Tweet

Lesson 101

// 102

// 103

// 104

---

lesson 105 — كلاس Calculator

lesson 106

---

lesson 107 — عمل واجهة باطاطا

```
import javax.swing.JFram;
```

---

lesson 108 — كلاس operating system

// 109

// 110

// 111

// 112

// 113

---

lesson 114 — نقطة مهمة جدا موضوع polymorphism

---

lesson 115 — ختام الدورة 102

^^